

# Containerization for R&D Applications

Daniel Trahan

*Daniel.Trahan@colorado.edu*

Andrew Monaghan

*Andrew.Monaghan@colorado.edu*

Please sign in:

<https://tinyurl.com/y4aks9zs>

Slides and exercises available for download at:

[https://github.com/ResearchComputing/NOAA\\_CONTAINER\\_TUTORIAL](https://github.com/ResearchComputing/NOAA_CONTAINER_TUTORIAL)  
SPRING\_2020



# Outline

## Part 1: Container fundamentals and Docker (1:00p-2:30p)

- Introduction to containers
- Docker commands and options
- Hands-on: Running Docker containers on your personal Machine
- Hands-on: Building Docker images
- Hands-on: Practical Application

## Part 2: Containers for HPC with Singularity (2:45p-3:30p)

- Singularity commands and options
- Hands-on: Running containers
- Building containers
- Special cases: Running containers for MPI and GPU jobs



# Introduction to Containers



# What is a container?

A container is a portable environment that packages some or all of the following: an operating system, software, libraries, compilers, data and workflows. Containers enable:

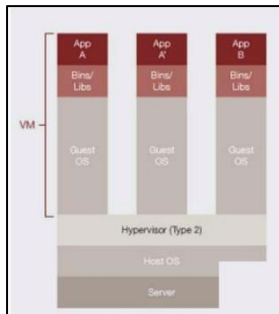
- Mobility of Compute
- Reproducibility (software and data)
- User Freedom



# Virtualization (1)

## Hardware virtualization (not used by containers!)

- Can run many OS's on same hardware (machine)
- E.g., VirtualBox, VMWare



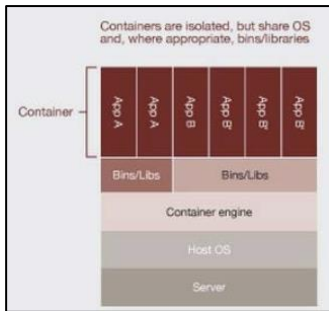
Material courtesy: M. Cuma, U. Utah

# Virtualization (2)

## OS-level virtualization (used by containers!)



- Can run many isolated OS instances (guests) under a server OS (host)
- Also called containers
- E.g., Docker, Singularity

*Best of both worlds: isolated environment that user wants, but can leverage host OS resources (network, I/O partitions, etc.)*



Material courtesy: M. Cuma, U. Utah

# Containerization software

- Docker 
  - Part 1 focus of today's tutorial
  - Well established – largest user base
  - Has Docker Hub for container sharing
  - Problematic with HPC
- Singularity 
  - Part 3 focus of today's tutorial
  - Designed for HPC
- Charliecloud; Shifter
  - Designed for HPC
  - Based on Docker
  - Less user-friendly

# Installing Docker

- Docker Community Edition
  - Windows: Windows 10 Professional or Enterprise
  - Mac: OS X El Capitan 10.11 or later
  - Linux
- Docker toolbox
  - Legacy solution for Windows and Mac for versions that do not meet the version requirements.
  - Utilizes the Virtual Box hypervisor for virtualization
  - For this tutorial, Windows users regardless of version will use Docker toolbox





# Why Docker?

- Probably the most popular containerization software
- Offers a variety of prebuilt images including:
  - Python
  - Perl
  - NodeJS
  - Ubuntu
- Very well documented with a large community creating and supporting docker images.
- DockerHub



# Docker Nuts and Bolts

- Docker runs on a concept of images and containers.
  - **Images**: Saved snapshots of a container environment.
    - Made from Dockerfile or pulled from Docker Hub
    - Stored in the Docker cache on your disk
  - **Containers**: Instances of images that are generated by Docker when an image is 'run'
    - Instance of image running in memory
    - Ephemeral Instances that cannot be continued
    - Can be run interactively

# Docker Commands

- Docker Commands are usually in the form of:  
`docker <sub-command> <flags> <target/command>`
- Examples:  
`docker run -it myimage`  
`docker container ls`  
`docker image prune`

# Running Docker Containers

- Run a docker image as a container:

```
docker run <image-name>
```

- Run a docker image interactively:

```
docker run -it <image-name>
```

- If an image is not on the system, then Docker will search Dockerhub to see if the image exists.
- Specify commands after your image to execute specific software in your container.

```
docker run <image-name> <program>
```

# Containerized Hello World

- Let's start with something simple:
  - Docker "Hello, World!"
    - Relatively small image
    - No dependencies
    - Built as a general test case
- Command:

```
docker run hello-world
```

# Docker Image and Container Commands

Image Commands	
<code>docker image ls</code>	List docker images stored in cache:
<code>docker image rm &lt;image&gt;</code> <code>docker rmi &lt;image&gt;</code>	Remove (an) image(s):
<code>docker image prune</code>	Remove unused images
Container Commands	
<code>docker container ls</code>	List docker containers currently running:
<code>docker container rm &lt;container&gt;</code> <code>docker rm &lt;container&gt;</code>	Remove (an) container(s):
<code>docker container prune</code>	Remove all stopped containers

# Docker Utility Commands

Commands	
<code>docker info</code>	Shows Docker system-wide information
<code>docker inspect &lt;docker-object&gt;</code>	Shows low-level information about an object
<code>docker config &lt;sub-command&gt;</code>	Manage docker configurations
<code>docker stats &lt;container&gt;</code>	Shows container resource usage
<code>docker top &lt;container&gt;</code>	Shows running processes of a container
<code>docker version</code>	Shows docker version information

- More details and commands can be found [on the docker documentation page](#)

# Demo 1: Running Containerized Python





# Demo 1: Python

- Running the Python docker container will pull Python from Docker Hub:

```
docker run python:3.7.2-slim
```

- ...did it work?
- Run your python image interactively:

```
docker run -it python:3.7.2-slim
```

- This puts us into a python interpreter, where you can run python code containerized in its own environment.

# Building Docker Containers

- To build a docker container, we need a set of instructions Docker can use to set up the environment.
  - Dockerfile
- Once we set up our dockerfile we can use the command  
`docker build -t <image-name> .`
- Then we can run the image with our `docker run` command  
`docker run <image-name>`

# Demo 2: Building an Ubuntu Container



# Demo 2: Setup

Dockerfiles and test files are provided for this workshop. We can pull the files from a github repository as such:

1. Navigate to your home directory

```
cd ~
```

2. Clone the repository

```
git clone
```

```
https://github.com/ResearchComputing/NOAA\_CONTAINER\_TUTORIAL\_SPRING\_2020
```

# Demo 2: Ubuntu w/ GCC

- For this first example we will be building a custom Ubuntu image that will provide a location to run the GNU Compiler Collection.
- Dockerfile provided
- Need to build:
  1. Navigate to the directory:  
`cd ~/Container_Tutorial_Fall_2019 /dockerdemo/ubuntu-gcc`
  2. Build the docker image with:  
`docker build -t happy-gcc .`
  3. Run the docker image as a container:  
`docker run -it happy-gcc`

# Editing Docker Images

- Suppose you have an existing docker image and want to make changes...
  - Rebuild Dockerfile!
    - Usually a bit cumbersome
    - No Dockerfile?
- Use docker commit!  
`docker run -it <image-name> bash` #or any shell...
- Then commit it to the image  
`docker commit <image-id> <image>`

# Mounting and Accessing files

- So now that we have a working container, how can we access the test files we downloaded?
  - Mounting directories: Bind Mount
    - Allows the docker container to access files on the host OS
    - Choose host's **source directory**, files in the directory will be moved to the container's **target directory**
      - **Source Directory**: Directory on the host system.  
Never within a container.
      - **Target Directory**: Directory in the Docker Container.  
Never on the host system.
    - A flag set within the **docker run** command:

```
docker run --mount type=bind,source=<source>,target=<target> <image>
```

# Mounting and Accessing files

- Mounting directories: Volume Mount
  - Same concept, but volumes are stored within docker cache.
  - Create Docker volumes in your terminal and link your volume directory
  - Similarly linked through the `docker run` command.

```
docker run --mount type=volume,source=<volume>,target=<target> <image>
```



# Demo 2 (Cont.): Mounting

- Returning to our demo, can we give our container access to our test files?
- Let's use a bind mount!
- In the directory where our Dockerfile lives... use this command (all on one line):

```
docker run -it --mount  
type=bind,source=$(pwd)/source,target=/target happy-gcc
```

- We can `cd /target` and run our test files!
- Command:

```
gcc hello.c -o hello.exe  
./hello.exe
```

# Demo 3: NCL container

- For this next example we will building a Docker image that will run the NCAR Command Language (NCL).
- Dockerfile provided
- Same process as before:
  1. Navigate to the Dockerfile found at:  
[~/Container\\_Tutorial\\_Fall\\_2019/dockerdemo/ncl](#)
  2. Build your docker file as an image titled "bright-ncl"
  3. Run your docker image as a container
- Can we test a sample script?

# Dockerhub

- The place where containers live!
- Dockerhub is a Docker hosted library of public and private Docker images.
  - Free and unlimited public images
  - 1 free private repository
- Great for hosting images for fellow researchers
- Commands similar to git

# Dockerhub Commands

- Download and upload docker images with ease.
  - `docker run <image>`
  - `docker pull <image>`
- Uploading a little more complicated...
  - Sign in with:  
`docker login`
  - List docker images with:  
`docker image ls`
  - Tag your image:  
`docker tag <image-id> <your-username>/<image-name>:<tag>`
  - Push!  
`docker push <your-username>/<image-name>`

# Docker-Compose

- Utility that can create and install docker images.
- Builds docker images based on a docker-compose.yml file.
  - YAML: *YAML Ain't Markup Language*
  - Data serialization language
  - Describes containers you wish to build with what features.
- Not a docker command!

# Docker-Compose Commands

- Build all containers in YAML file:  
`docker-compose build`
- Build and run all containers in YAML file:  
`docker-compose up`
- List all containers in YAML file:  
`docker-compose images`
- Run a one-off command from a container:  
`docker-compose run <container-name> <command>`

# Part 2: Containers for HPC with Singularity



[https://github.com/ResearchComputing/Container\\_tutorial\\_Spring\\_2020](https://github.com/ResearchComputing/Container_tutorial_Spring_2020)

# Part 2 Outline

- Singularity commands and options
- Hands-on: Running containers
- Building containers





# Why Singularity?

- Singularity is a comparably safe container solution for HPC
  - User is same inside/outside container
  - User cannot escalate permissions without administrative privilege
- Can support MPI and GPU resources on HPC (scaling)
- Can use HPC filesystems
- Supports the use of Docker containers
- Container is seen as a file, and can be operated on as a file



# Singularity Overview

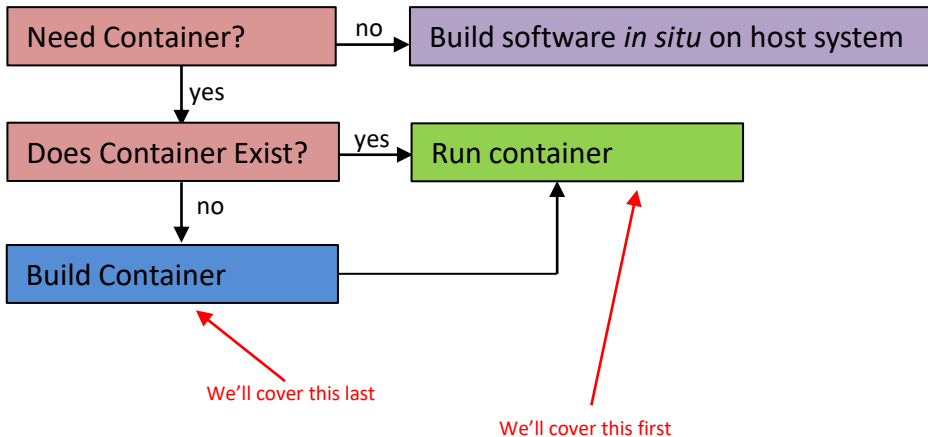
Singularity Workflow

Key Commands

Running Containers



# The Singularity Workflow



# Where do I find existing containers?

Docker Hub: <https://hub.docker.com>

Singularity Hub: <https://singularity-hub.org>

Sylabs Library: <https://cloud.sylabs.io/library>

Tutorial on finding a running an existing Docker container:

<https://www.chpc.utah.edu/documentation/software/singularity.php#exd>

# Key Singularity Commands

build: Build a container on your user endpoint or build environment

exec: Execute a command to your container

inspect: See labels, run and test scripts, and environment variables

pull: pull an image from Docker or Singularity Hub

run: Run your image as an executable

shell: Shell into your image

*More: <https://www.sylabs.io/guides/3.2/user-guide/cli.html>*

# Running containers

Now, on **any system** with Singularity, even without administrative privilege, you can retrieve and use containers:

- Download a container from Singularity Hub or Docker Hub
  - `singularity pull shub://some_repo/some_image`
  - `singularity pull library://some_repo/some_image`
  - `singularity pull docker://some_repo/some_image`
- Run a container
  - `singularity run mycont.sif`
- Execute a specific program within a container
  - `singularity exec mycont.sif python myscript.py`
- “Shell” into a container to use or look around
  - `singularity shell mycont.sif`
- Inspect an image
  - `singularity inspect --runscript mycont.sif`

# Running Containers

Navigate to the tutorial:  
<https://bit.ly/3etaDXD>



*The tutorial can also be found here:*  
[https://github.com/ResearchComputing/Container\\_tutorial\\_Spring\\_2020](https://github.com/ResearchComputing/Container_tutorial_Spring_2020)

# Building containers





# There are 3 ways to build a Singularity container

1. Build a container on a system on which you have administrative privilege (e.g., your laptop).
  - **Pros:** You can interactively develop the container.
  - **Cons:** Requires many GB of disk space, requires administrative privilege, must keep software up-to-date, container transfer speeds can be slow depending on personal network connection.
2. Build a container on Singularity Hub using recipe, Github
  - **Pros:** Essentially zero disk space required on your system, doesn't require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, typically faster transfers from Singularity Hub to desired endpoint.
  - **Cons:** Cannot interactively develop the container
3. Build a container on Sylabs remote builder
  - **Pros:** Essentially zero disk space required on your system, doesn't require administrative privilege, no software upgrades needed, easy to retrieve from anywhere, final container is placed on local machine
  - **Cons:** Cannot interactively develop the container, 'Freemium' version limited

# What is a recipe?

Header

```
Bootstrap:docker
From:ubuntu:latest
```

Metadata

```
%labels
MAINTAINER Andy M
```

Runtime  
environment  
variables

```
%environment
HELLO_BASE=/code
export HELLO_BASE
```

Default program  
at runtime

```
%runscript
echo "This is run when you run the image!"
exec /bin/bash /code/hello.sh "$@"
```

Where software  
and directories  
are installed at  
buildtime

```
%post
echo "This section is performed after you bootstrap to build the image."
mkdir -p /code
apt-get install -y vim
echo "echo Hello World" >> /code/hello.sh
chmod u+x /code/hello.sh
```

More: [https://www.sylabs.io/guides/3.2/user-guide/definition\\_files.html](https://www.sylabs.io/guides/3.2/user-guide/definition_files.html)

# Container Formats

\*.sif format and older \*.sing format  
(immutable final container format)

- **squashfs**: the default container format is a compressed read-only file system that is widely used for things like live CDs/USBs and cell phone OS's
- **ext3**: (also called **writable**) a writable image file containing an ext3 file system that was the default container format prior to Singularity version 2.4
- **directory**: (also called **sandbox**) standard Unix directory containing a root container image
- **tar.gz**: zlib compressed tar archive
- **tar.bz2**: bzip2 compressed tar archive
- **tar**: uncompressed tar archive

Writeable sandbox used for interactive container development

<https://singularity.lbl.gov>

# 1. Building a container interactively (demo)

- Bootstrap a base container (has OS you want, maybe other stuff too) into a sandbox:

```
sudo singularity build --sandbox mytest/ docker://alpine:latest
```

- Shell into the container and install what you need by trial and error:

```
sudo singularity shell --writable mytest/
```

- [now do stuff in container; as you get it correct, add commands to Singularity recipe]

- Now finalize container.

- You can either build squashfs image from sandbox:

```
sudo singularity build mytest.sif mytest/
```

- Or you can build squashfs image from recipe (best practice):

```
sudo singularity build mytest.sif Singularity
```

## 2. Building a container on Singularity Hub (basic steps)

1. Create a recipe file for your container
2. Name it "Singularity"
3. Create a github repository for your container
4. Upload it to your github repository
5. Log into Singularity Hub using your github username/password
6. Go to "My Collections" and choose "ADD A COLLECTION"
7. Select the github repository you just uploaded.
8. The container will build automatically.
9. Revised recipes are automatically rebuilt when pushed to github.
10. Additional details at:

<https://github.com/singularityhub/singularityhub.github.io/wiki>

*\*\*Note: You can build Docker containers on Docker Hub using a nearly identical process!*

# 3. Building a container with Sylabs Remote Builder (demo)

1. Get a token from Sylabs Cloud: <https://cloud.sylabs.io/auth>
  1. Login using your Github account
  2. Provide a label under "Create a New Access Token"
  3. Click "Create New Token"
  4. Copy the token string (<your-token>)
2. Add the token on your host machine:
  1. `mkdir ~/.singularity`
  2. `echo "<your-token>" > ~/.singularity/sylabs-token`
3. Issue the command to build your container remotely
  1. `singularity build --remote myimage.sif myrecipe.def`
4. If successful, the container will be placed in your working directory

# Thank you!

Please fill out the survey:

<http://tinyurl.com/curc-survey18>

Contact information:

[Andrew.Monaghan@Colorado.edu](mailto:Andrew.Monaghan@Colorado.edu); [Daniel.Trahan@Colorado.edu](mailto:Daniel.Trahan@Colorado.edu)

Additional learning resources:

*Slides and Examples from this course:*

[https://github.com/ResearchComputing/NOAA\\_CONTAINER\\_TUTORIAL\\_SPRING\\_2020](https://github.com/ResearchComputing/NOAA_CONTAINER_TUTORIAL_SPRING_2020)

*Web resources:*

<https://training.play-with-docker.com> (docker online training materials)

<https://hub.docker.com> (Docker Hub)

<https://www.sylabs.io/guides/3.4/user-guide/> (user guide for Singularity)

<https://www.singularity-hub.org/> (Singularity Hub)

# Extra slides





# Hands-on Example

- Build a container on Singularity Hub



# Go to the example directory

Go to the Singularity Hub example directory on RMACC Summit

```
cd /scratch/summit/$USER/  
git clone https://github.com/ResearchComputing/Container\_Tutorial\_Fall\_2019  
Cd Container_Tutorial_Fall_2019/build_container_on_shub/
```

List the contents of the directory and see a description of each file:

```
ls
```

If you get behind or have issues, look in 'tutorial\_steps.txt'

```
cat tutorial_steps.txt
```

# Prepare and upload your recipe

Use a text editor to explore and customize the recipe file 'Singularity':

```
nano Singularity
```

- Can you determine the purpose of each section?
- Can you determine what this container does?
- Are there any files copied to the container?
- Now change the 'Maintainer' to your name
- To exit and save type [ctrl-x], then "y", then [enter].

Now we are ready to build a container on Singularity Hub using this recipe. Recall that we do this by creating a github repository containing the 'Singularity' recipe file. For the sake of time we've created a script to facilitate this step. Type:

```
./make_git_repo.sh <your-github-username>
```

You will be prompted for your github password 2 times while the script is running; enter it each time. Use your browser to confirm your repository was created:

```
https://github.com/<your-github-username>/build\_container\_on\_shub
```

# Build your container on 'shub'

Now navigate to Singularity Hub in your browser:

<https://www.singularity-hub.org/>

...and do the following:

1. Go to the "login" link in the upper right corner and login with your github credentials
2. Choose "GITHUB", not "GITHUB (WITH PRIVATE)"
3. Go to "My container collections"
4. Go to "Add a Collection"
5. Choose "<your-github-username>/build\_container\_on\_shub" and click on "SUBMIT"
6. This will take you another page while your container builds. You can click "refresh" to check it's status. It may take several minutes.
7. If your container builds successfully, you will see a green 'Complete' button. If not, let us know and we'll help: you can quickly fix and re-commit the recipe to github, which will initiate another build on Singularity Hub.

# Now pull your container from 'shub' and run it!

Now navigate to Singularity Hub in your browser:

```
singularity pull --name mytranslator.sif shub://<your-github-username>/build_container_on_shub
```

Run your container with the 'inside' version of the python script:

```
singularity run mytranslator.sif
```

...and run your container with the 'outside' version of the python script:

```
singularity exec mytranslator.sif python ./text_translate.py
```

(hint: you can change the language in ./text\_translate.py to confirm that the outside script is running)

# Notes: MPI and GPUs



# Notes: MPI-enabled containers

HPC systems use low-latency interconnects (“fabric”) to enable MPI to be efficiently implemented across nodes). In order to use a Singularity container with OpenMPI (or any MPI) on a Supercomputer there are two requirements:

1. The Singularity container needs to have the fabric libraries installed inside.
2. OpenMPI needs to be installed both inside and outside of the Singularity container. More specifically, the SAME version of OpenMPI needs to be installed inside and outside (at least very similar, you can sometimes have two different minor versions, ex: 2.1 and 2.0).

# Running containers on GPUs

Syntax (after loading Singularity on a gpu node):

```
singularity exec --nv docker://tensorflow/tensorflow:latest-gpu  
python mytensorflow_script.py
```

With the **--nv** option the driver libraries do not have to be installed in the container. Instead, they are located on the host system (e.g., Teton) and then bind mounted into the container at runtime. This means you can run your container on a host with one version of the NVIDIA driver, and then move the same container to another host with a different version of the NVIDIA driver and both will work. (Assuming the CUDA version installed in your container is compatible with both drivers.)

The NVIDIA Driver version on a GPU node can be queried (when on that node) with the command **nvidia-smi**. See <https://docs.nvidia.com/deploy/cuda-compatibility/>

You can build a container by bootstrapping a base NVIDIA CUDA image (with a compatible CUDA version) from: <https://hub.docker.com/r/nvidia/cuda/>

NVIDIA has a tool for building gpu-enabled containers for both Singularity and Docker. See: <https://github.com/NVIDIA/hpc-container-maker>



# Troubleshooting and Caveats



# Troubleshooting (1)

## Container/host environment conflicts

- Container problems are often linked with how the container “sees” the host system. Common issues:
  - The container doesn't have a bind point to a directory you need to read from / write to
  - The container will “see” python libraries installed in your home directory (and perhaps the same is true for R and other packages. If this happens, set the PYTHONPATH environment variable in your job script so that it points to the container paths first.
    - `export PYTHONPATH=<path-to-container-libs>:$PYTHONPATH`
- To diagnose the issues noted above, as well as others, “shelling in” to the container is a great way to see what's going on inside. Also, look in the singularity.conf file for system settings (can't modify).

# Troubleshooting (2)

Failures during container pulls that are attributed (in the error messages) to `*tar.gz` files are often due to corrupt `tar.gz` files that are downloaded while the image is being built from layers. Removing the offending `tar.gz` file will often solve the problem.

When building ubuntu containers, failures during `%post` stage of container builds from a recipe file can often be remedied by starting the `%post` section with the command `apt-get update`. As a best practice, make sure you insert this line at the beginning of the `%post` section in all recipe files for ubuntu containers.

<https://singularity.lbl.gov>



# Caveats (1)

We didn't cover overlays. These are additional images that are "laid" on top of existing images, enabling the user to modify a container environment without modifying the actual container. Useful because:

1. Overlay images enable users to modify a container environment even if they don't have root access (though changes disappear after session)
2. Root users can permanently modify overlay images without modifying the underlying image.
3. Overlays are a likely way to customize images for different HPC environments without changing the underlying images.
4. More on overlays: <http://singularity.lbl.gov/docs-overlay>

We didn't cover GPU usage with containers. See:

<http://singularity.lbl.gov/archive/docs/v2-3/tutorial-gpu-drivers-open-mpi-mtls>

# Caveats (2): Moving containers

You've built your first container on your laptop. It is 3 Gigabytes. Now you want to move it to Summit to take advantage of the HPC resources. What's the best way?

Remember, containers are files, so you can transfer them to Summit just as you would a file:

- Command line utilities (scp, sftp)
- **Globus** (recommended)
- For more on data transfers to/from Summit:  
<https://github.com/ResearchComputing/Research-Computing-User-Tutorials/wiki/Data-Transfers>